图文 043、动手实验：自己动手模拟出频繁Young GC的场景体验一下

1895 人次阅读 2019-08-12 07:00:00

详情 评论

动手实验：

自己动手模拟出频繁Young GC的场景体验一下！

给大家推荐一套质量极高的Java面试训练营课程：

作者是中华石杉，石杉老哥是我之前所在团队的 Leader ，骨灰级的技术神牛！

大家可以点击下方链接，了解更多详情，并进行试听：

21天互联网Java进阶面试训练营（分布式篇）

1、前文回顾

从本周开始，我们将要全面进入实操环节，也就是说之前几周时间都是在分析JVM的运行原理、GC原理以及优化原理，但是本周开始我们将要通过各种代码模拟出来JVM的各种场景，同时结合GC日志去分析到底JVM是怎么运行的。

今天的文章，我们将会给大家通过代码演示年轻代的Young GC是如何发生的，同时告诉大家如何在JVM参数中去配置打印对应的GC日志，然后我们通过GC日志来慢慢的分析JVM的GC到底是如何运行的。

2、程序的JVM参数示范

首先，我们通过之前的学习都知道，我们平时系统运行创建的对象，除非是那种大对象，否则通常来说都是优先分配在新生代中的Eden区域的

而且新生代还有另外两块Survivor区域，默认Eden区域占据新生代的80%，每块Survivor区域占据新生代的10%。

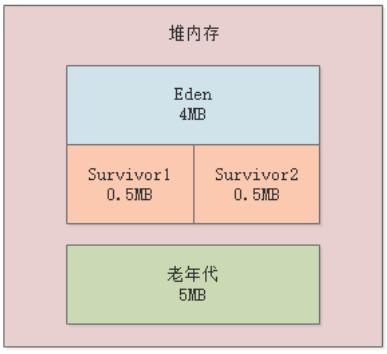
比如我们用以下JVM参数来运行代码：

-XX:NewSize=5242880 -XX:MaxNewSize=5242880 -XX:InitialHeapSize=10485760 -XX:MaxHeapSize=10485760 -XX:SurvivorRatio=8 -XX:PretenureSizeThreshold=10485760 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC

上述参数都是基于JDK 1.8版本来设置的，不同的JDK版本对应的参数名称是不太一样的，但是基本意思是类似的。

上面“-XX:InitialHeapSize”和“-XX:MaxHeapSize”就是初始堆大小和最大堆大小，“-XX:NewSize”和“-XX:MaxNewSize”是初始新生代大小和最大新生代大小，“-XX:PretenureSizeThreshold=10485760”指定了大对象阈值是10MB。

相当于给堆内存分配10MB内存空间，其中新生代是5MB内存空间，其中Eden区占4MB，每个Survivor区占0.5MB，大对象必须超过10MB才会直接进入老年代，年轻代使用ParNew垃圾回收器，老年代使用CMS垃圾回收器，看下图图示。



3、如何打印出JVM GC日志?

接着我们需要在系统的JVM参数中加入GC日志的打印选型，如下所示：

-XX:+PrintGCDetils：打印详细的gc日志
-XX:+PrintGCTimeStamps：这个参数可以打印出来每次GC发生的时间
-Xloggc:gc.log：这个参数可以设置将gc日志写入一个磁盘文件

加上这个参数之后，jvm参数如下所示：

-XX:NewSize=5242880 -XX:MaxNewSize=5242880 -XX:InitialHeapSize=10485760 -XX:MaxHeapSize=10485760 -XX:SurvivorRatio=8 -XX:PretenureSizeThreshold=10485760 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:gc.log

4、示例程序代码

接着我们给大家看一段示例程序代码：

```
public class Demo1 {

    public static void main(String[] args) {
        byte[] array1 = new byte[1024 * 1024];
        array1 = new byte[1024 * 1024];
        array1 = new byte[1024 * 1024];
        array1 = null;

        byte[] array2 = new byte[2 * 1024 * 1024];
    }

}
```

5、对象是如何分配在Eden区内的

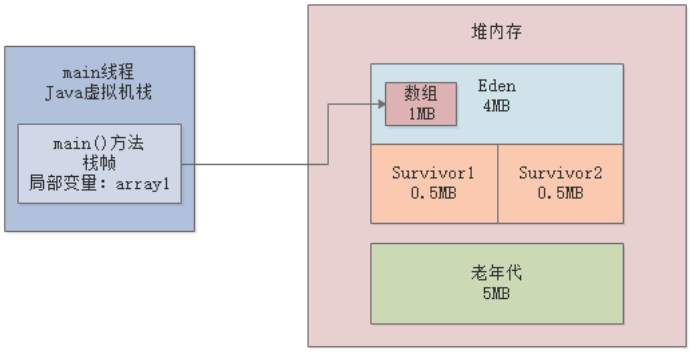
上面的这段代码非常简单，先通过“new byte[1024 * 1024]”这样的代码连续分配了3个数组，每个数组都是1MB

然后通过array1这个局部变量依次引用这三个对象，最后还把array1这个局部变量指向了null

那么在JVM中上述代码是如何运行的呢?

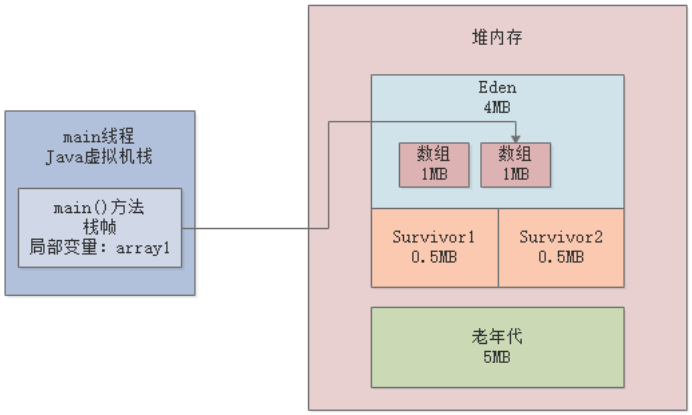
首先我们来看第一行代码：byte[] array1 = new byte[1024 * 1024];。

这行代码一旦运行，就会在JVM的Eden区内放入一个1MB的对象，同时在main线程的虚拟机栈中会压入一个main()方法的栈帧，在main()方法的栈帧内部，会有一个“array1”变量，这个变量是指向堆内存Eden区的那个1MB的数组，如下图。



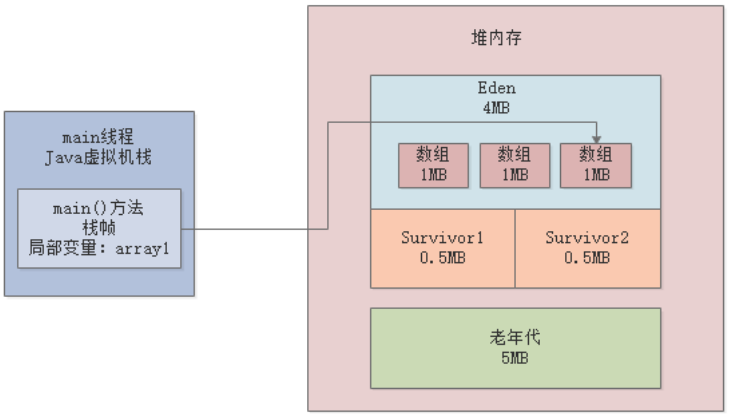
接着我们看第二行代码：array1 = new byte[1024 * 1024];

此时会在堆内存的Eden区中创建第二个数组，并且让局部变量指向第二个数组，然后第一个数组就没人引用了，此时第一个数组就成了没人引用的“垃圾对象”了，如下图所示。



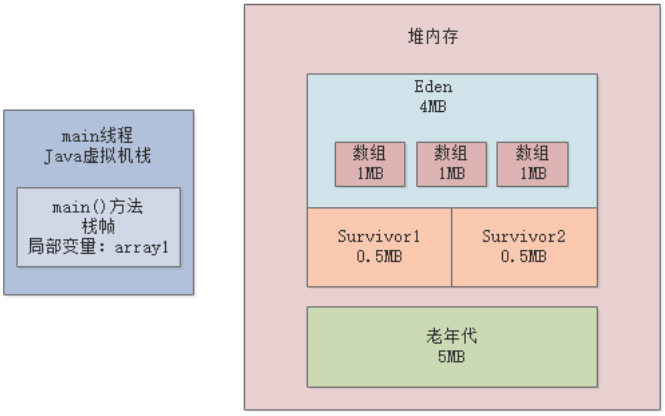
然后看第三行代码：byte[] array1 = new byte[1024 * 1024];。

这行代码在堆内存的Eden区内创建了第三个数组，同时让array1变量指向了第三个数组，此时前面两个数组都没有人引用了，就都成了垃圾对象，如下图所示。



接着我们来看第四行代码：array1 = null;。

这行代码一执行，就让array1这个变量什么都不指向了，此时会导致之前创建的3个数组全部变成垃圾对象，如下图。



最后看第五行代码：byte[] array2 = new byte[2 * 1024 * 1024];。

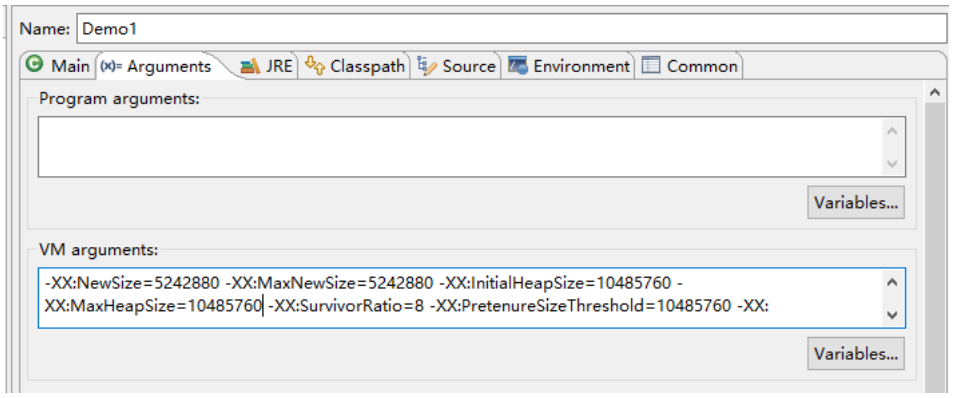
此时会分配一个2MB大小的数组，尝试放入Eden区中，大家觉得这个时候Eden区能放的下吗？

明显是不行的，因为Eden区总共就4MB大小，而且里面已经放入了3个1MB的数组了，所以剩余空间只有1MB了，此时你放一个2MB的数组是放不下的。

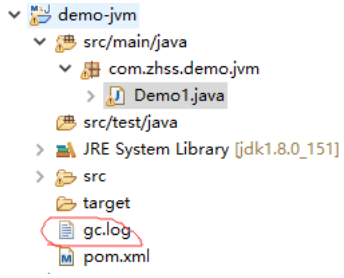
所以这个时候就会触发年轻代的Young GC。

6、采用指定JVM参数运行程序

之前给大家讲过，在Eclipse等开发工具里如何以指定JVM参数运行程序，就是对你的程序右键，然后选择“Run As -> Run Configurations”，接着在下图中填入对应的JVM参数：



然后运行即可，此时运行完毕后，会在下述工程目录中出现一个gc.log文件，里面就是本次程序运行的gc日志，如下图所示。



打开gc.log文件，我们会看到如下所示的gc日志：

Java HotSpot(TM) 64-Bit Server VM (25.151-b12) for windows-amd64 JRE (1.8.0_151-b12), built on Sep 5 2017 19:33:46 by "java_re" with MS VC++ 10.0 (VS2010)

Memory: 4k page, physical 33450456k(25709200k free), swap 38431192k(29814656k free)

CommandLine flags: -XX:InitialHeapSize=10485760 -XX:MaxHeapSize=10485760 -XX:MaxNewSize=5242880 -XX:NewSize=5242880 -XX:OldPLABSize=16 -XX:PretenureSizeThreshold=10485760 -XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:SurvivorRatio=8 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseConcMarkSweepGC -XX:-UseLargePagesIndividualAllocation -XX:+UseParNewGC

0.268: [GC (Allocation Failure) 0.269: [ParNew: 4030K->512K(4608K), 0.0015734 secs] 4030K->574K(9728K), 0.0017518 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

Heap

par new generation total 4608K, used 2601K [0x00000000ff600000, 0x00000000ffb00000, 0x00000000ffb00000)

eden space 4096K, 51% used [0x00000000ff600000, 0x00000000ff80a558, 0x00000000ffa00000)

from space 512K, 100% used [0x00000000ffa80000, 0x00000000ffb00000, 0x00000000ffb00000)

to space 512K, 0% used [0x00000000ffa00000, 0x00000000ffa00000, 0x00000000ffa80000)

concurrent mark-sweep generation total 5120K, used 62K [0x00000000ffb00000, 0x0000000100000000, 0x0000000100000000)

Metaspace used 2782K, capacity 4486K, committed 4864K, reserved 1056768K

class space used 300K, capacity 386K, committed 512K, reserved 1048576K

是不是觉得乱七八糟，密密麻麻的?

没关系，明天的文章我们会对照gc日志以及通过一步一图的方式，来分析一下到底Young GC是如何运行的。

7、今日思考题

给大家今天留一个小的思考题，自己结合之前学习过的JVM原理，包括我们已经画出来的图，还有GC日志，分析一下，这次Young GC到底是如何运行的?